항공우주산업기술동향 2권1호 (2004) pp. 86~93
기술동향
www.kari.re.kr
/lib/index.html
에서 보실 수 있습니다.

실시간 소프트웨어 개발기술 동향

글 / 구 철 회 chkoo@kari.re.kr
한국항공우주연구원 통신해양기상위성사업단
통신위성체계그룹

1. 서론

현재에는 개인용 컴퓨터도 전화기와 마찬가지로 가정에 많이 보급이 되어 있으며 성능 또한 개인용 컴퓨터란 말이 무색할 정도로 해를 거듭할수록 크게 개선되고 있다. 처음 컴퓨터가 개발되었을 때 컴퓨터의 성능은 주로 하드웨어의 성능에 따라서 평가되었고 하드웨어에 대한 개발이 주였지만 1970년대 중반 이후부터는 소프트웨어의 성능이 컴퓨터의 성능을 좌우하게 되었다. 대부분의 컴퓨터 산업 비용이 소프트웨어 부분에서 발생하게 되고 컴퓨터 산업의 부가가치 창출에서 소프트웨어가 차지하는 비중이 높아지게 되었다.

현재에 이르러 개인용 컴퓨터의 소프트웨어는 개인의 삶의 질을 크게 향상시켜 주고 복잡한 서류 작업과 가게 매출 정산 작업을 쉽게 만들어 주고 있다. 또 공장에 보급된 생산 자동화 기계를 제어하는 소프트웨어는 안전하고도 정확한 제품을 생산하는데 기여하고 있다.

데스크톱(Desk Top) 소프트웨어나 인터넷 소프트웨어를 범용 소프트웨어로 분류한다면 자동화 기계의 소프트웨어는 내장형 소프트웨어(Embedded Software)로 분류할 수 있다.

소프트웨어 개발 요구사항의 관점에서 범용 소프트웨어와 내장형 소프트웨어를 비교하여 보면, 범용 소프트웨어는 소프트웨어의 사용범위가 일반인으로 폭넓은 사용자에게 보다 나은 서비스를 제공하기 위한 개발 요구사항을 가진다고 할 수 있다. 반면 내장형 소프트웨어는 매우 제한된 사용자를 대상으로 하고 있으며 정확한 동작 보장과 최소 비용을 목적으로 제작된다[9]. 위성의 탑재 소프트웨어도 내장형 소프트웨어의 범주에 포함된다고 할 수 있다.

범용 소프트웨어와 내장형 소프트웨어는 소프트웨어의 적용 대상과 사용자를 기준으로 분류한 것인데 소프트웨어를 실시간(Real Time)과 비실시간(Non Real Time) 으로 분류하는 것도 가능하다.

인공위성용 탑재 소프트웨어와 같이 실시간 특성을 만족해야 하는 내장형 소프트웨어의 경우에는 실시간 요구사항을 만족해야 하는 경우가 발생할 수 있다. 실시간 특성의 요구사항을 만족시키지 못하는 소프트웨어는 1997년 화성에 도착한 화성 탐사선 "Mars Pathfinder"의 임무 실패에도 알 수 있듯이 심각한 재산 피해는 물론이고 심지어는 인명 피해도 불러올 수 있다.

본 논문에서 실시간 소프트웨어의 기본 개념을 설명하고 현재의 기술 동향을 소개함으로써 위성 탑재 소프트웨어를 비롯한 내장형 실시간 소프트웨어 개발에 도움이 되기를 희망한다.

2. 실시간 및 비실시간

일반적인 소프트웨어 분류에서 소프트웨어의 성격을 실시간과 비실시간으로 구분하는 것은 매우 중대한 의미를 갖는다. 왜냐하면 이 분류에 따라서 소프트웨어의 특성은 전혀 다른 것이 되기 때문이다.

먼저 비실시간 소프트웨어를 살펴보면 비실시간 소프트웨어는 소프트웨어 연산의 정확성에 주요한 의미를 둔다. 예를 들어 개인용 컴퓨터로 음악을 듣고 있을 때 사용자가 문서 편집 소프트웨어를

구동시켰다면 약간 시간 지연을 가지고 소프트웨어가 구동될 것이지만 이것에 대해서 심하게 불평하는 사용자는 그리 많지 않을 뿐 아니라 피해는 거의 없을 것이다. 또한 개인용 컴퓨터로 영화를 감상할 때에도 영화가 중간에 아주 짧은 시간(0.1 s ~ 1.0 s 정도) 정도 끊긴다고 하더라도, 심지어 여러번 그러한 일이 발생한다고 하더라도 큰 문제는 발생하지 않는다.

반면 실시간 소프트웨어는 소프트웨어 연산의 정확한 결과뿐만 아니라 소프트웨어가 정확한 결과를 출력하는데 걸리는 시간이 매우 중요한 의미를 갖는다. 실시간의 의미는 매우 중요하므로 아래와 같이 이 분야에서 자주 인용되는 실시간 개념을 원문으로 소개하였다.

“A real-time system is one in which the correctness of the computations not only depends upon the logical correctness of the computation but also upon the time at which the result is produced. If the timing constraints of the system are not met, system failure is said to have occurred[1].”

예를 들어 전투기의 경우 자기에게 날아오는 적기의 미사일을 회피하는 소프트웨어가 뒤늦게 회피 동작을 수행했다면 이미 전투기는 격추된 후일 것이다. 또 자동차를 조립하는 컨베어 벨트 상의 자동화 조립 기계들이 만약 정해진 시간을 지켜 작업하지 않는다면 엉뚱한 곳에 철판이 용접되는 사태가 벌어질 것이다.

실시간 개념은 실시간 요구사항의 중요성 정도에 따라 강성 실시간(Hard Real-Time)과 연성 실시간(Soft Real-Time)으로 분류할 수 있다. 강성 실시간 요구사항은 소프트웨어가 반드시 정해진 시간 내에 정확한 결과를 내기를 요구한다. 만일 정해진 시간 내에 결과를 내지 못했다면 설사 결과가 정확하더라도 실패로 간주한다. 하지만 연성 실시간 요구사항은 소프트웨어가 정해진 시간 내에 결과를 내지 못하더라도 어느 정도의 시간 지연은 용납한다. 대부분의 실시간 시스템은 연성 실시간 시스템으로 구현가능하다. 특별히 로봇 제어 또는 미사일 제어 등이 대표적인 강성 실시간 시스템이다. 인공위성의 경우 충분히 연성 실시간 시스템으로 구현이 가능할 것이다. 실제로 인공위성 자세제어 모듈의 경우 대략 8 Hz의 동작으로도 성공적으로 위성의 자세를 유지하는 것으로 확인되었다.

3. 실시간 소프트웨어 적용 분야

실시간 소프트웨어의 적용 분야는 생활 주변에서는 손쉽게 찾아볼 수 있다. 저가, 저소비전력, 고속의 마이크로프로세서가 구현이 가능해 짐에 따라 이전에 복잡했던 디지털 로직으로 구현되었던 기능이 현재는 마이크로프로세서를 사용한 내장형 시스템으로 전환되고 있다.

본 논문에서는 실시간 소프트웨어의 적용 분야를 강성 실시간 시스템과 연성 실시간 시스템 영역으로 구분해 보았으며 결과는 표 1과 같다.

일반 범용 소프트웨어의 기능은 입력된 데이터를 수학적 연산을 통해서 변환하고 그 결과를 출력하는 것이다. 하지만 실시간 소프트웨어의 목적은 실제 물리계에서 발생되는 현상을 처리하는 것이다. 예를 들어 로봇을 제어하는 소프트웨어의 경우 이 소프트웨어는 로봇의 각 관절 및 전자눈(Electronic Eyes)에서 나오는 수십 개의 센서 신호를 동시에 처리하여야 한다. 또한 로봇의 자세를 유지하기 위한 여러 개의 작동 모터(Actuator)를 제어해야 한다. 어느 한 부분도 원활하게 유기적으로 수행되지 않는다면 로봇은 생각한데로 동작해 주지 않을 것이다.

표 1. 실시간 소프트웨어의 응용 분야

	강성 실시간 시스템	연성 실시간 시스템
적용 분야	화학 공장 제어 원자력 발전소 제어 철도 제어 자동차 제어 비행기 제어 공장 자동화 로봇 공학 군용 장비	기상 관측 우주선 제어 가상 현실 가전 제품
예	비행기, 자동차, 로봇, 무기 통제,	전화기, 핸드폰, 가전제품, 장난감, 보안 장비, 텔레비전, 프린터, 스캐너, 기상 예보,인공위성

4. 실시간 소프트웨어 개발 도구

여러 개의 물리적 신호들을 동시에 처리해야 하는 실시간 시스템의 동시성(Concurrency)은 실시간 소프트웨어 이론의 가장 핵심이자 이해하기 곤란한 토픽중의 하나이다. 일반적인 범용 소프트웨어의 쓰레드와 프로세스 개념은 원시적이어서 복잡한 실시간 시스템을 설계하고 이해하는 데는 부족하다. 복잡한 실시간 시스템을 이해하고 설계하기 위해서는 근본적인 동시성 이론(Concurrency Theory)에 대한 공부가 필요하다.

4.1 RTOS가 필요한가?

RTOS(실시간 운영체제, Real-Time Operating System)는 실시간 소프트웨어를 개발할 때 흔히 사용을 고려하게 되는 OS이다.

이 논제에 대해서 논하기 위해서는 일단 RTOS를 실시간 커널(Real-Time Kernel)과 운영체제 유틸리티(OS Utility)로 구분하는 것이 필요할 것이다.

결론적으로 말한다면 실시간 커널은 반드시 필요하지는 않다고 할 수 있다[17]. 실시간 시스템의 특성은 매우 다양한 양상을 띠기 때문에 일부 실시간 시스템에서는 실시간 커널이 전혀 필요 없을 수가 있고 반대로 다른 실시간 시스템은 반드시 실시간 커널이 필요로 하는 경우도 있다. 또 실시간 커널이라고 해도 비선점형 커널(Non - preemptive Kernel)이 필요할 수도 있고 선점형 커널(Preemptive Kernel)이 필요할 수도 있다.

하지만 인터럽트 서비스 루틴(ISR), 메모리 관리(Memory Management), 타이머(Timer), 리소스 관리(Resource Management), 예외 처리(Exception Handling) 등과 같은 운영체제 유틸리티는 실시간 소프트웨어 개발에 매우 필요한 자원이라고 할 수 있다.

매우 작은 내장형 실시간 시스템을 제외한 중, 대형 실시간 시스템 개발에 있어서 위와 같은 운영체제 유틸리티가 없다면 개발의 진척은 크게 제약을 받게 될 것이다. 따라서 설계하고자 하는 실시간 시스템에 적당한 RTOS를 선택하고, 구현하고자 하는 실시간 시스템의 특성에 따라 실시간 커널 기능의 일부분 또는 모두를 사용하는 것이 타당하다고 생각한다. 커널기능의 일부분을 필요에 따라 제거하거나 추가할 수 있는 구성 가능한 커널(Scalable Kernel)의 사용은 일반적으로 성능(CPU Throughput 뿐만 메모리도 해당)이 제한된 내장형 시스템의 개발에 도움이 된다. 표 2에는 실시간 시스템의 규모에 따른 RTOS의 분류를 소개하였다[14].

표 2. 시스템 규모에 따른 RTOS 분류

	소규모	중규모	대규모
RTOS Example	PALOS TinyOS	uCOS-II eCos	VxWorks RTLinux

실시간 소프트웨어에서 스케줄링의 목적은 각 타스크마다 할당되어 있는 최종시한(Deadline)내에 모든 연산 기능의 수행이 완료되도록 보장하는 것이다. 일반적으로 각 타스크 실행에는 각각 실행 우선순위(Priority)가 할당되어 실행된다. 스케줄링은 비선점형 스케줄링과 선점형 스케줄링으로 나눌 수 있다.

스케줄링의 종류는 여러 가지이며 본 논문에서 이 모든 종류에 대해서 세세히 다루는 것은 적당하지 않으리라고 판단하여 표 3에 그 종류와 특징을 간략히 적었다. 개발하고자 하는 시스템의 특성을 파악한 후 적절한 스케줄링 방법을 사용하면 타당하리라 본다.

실시간 시스템의 특성상 실시간 소프트웨어의 테스트는 매우 어렵다. 기본적으로 스케줄링되는 타스크들의 수행은 다른 타스크와 리소스 또는 자극(Stimuli)에 따른 경쟁 조건(Race Condition)에 의해서 이루어지기 때문에 이전에 발생한 오류가 복잡한 실시간 시스템에서 다시 관측될 확률은 극히 적다[20].

5. 실시간 소프트웨어 개발 언어

실시간 시스템은 범용 소프트웨어와 달리 실제 물질세계의 현상을 처리하기 때문에 실시간 소프트

표 3. 스케줄링 방법의 종류와 특징

종 류	특 징
Function Queue Scheduling (Control Flow)	• 순차적 함수 실행 방식(FIFO Scheduling) • 비선점형 스케줄링 • 제어 순서에 따라서 제어 함수 실행 • 예외 처리에 주의하여야 함
Static Scheduling (Synchronous Data Flow)	• 데이터의 흐름에 따른 함수 실행 방식 • 입수되는 데이터를 담당하는 함수를 실행 • 완전히 동기화되거나 예측 가능한 데이터 통신이 보장되어야 함
Rate Monotonic Scheduling(RMS)	• 각 타스크를 각 타스크마다 정해진 주기에 맞추어 실행하는 방식 • 각 타스크의 주기에 대한 해석을 통해서 최종시한을 넘기는 타스크가 발생하지 않도록 우선순위를 할당함 • 빠른 주기의 타스크가 높은 우선순위를 가짐 • 타스크의 최종시한은 주기와 같거나 특수한 경우 주기보다 연장 가능 • 각 타스크간의 동기화는 허용되지 않을 수도 있음
Deadline Monotonic Scheduling(DMS)	• 각 타스크의 최종시한에 대한 해석을 통해서 최종시한을 넘기는 타스크가 발생하지 않도록 우선순위를 할당함 • 빠른 주기의 타스크가 높은 우선순위를 가짐 • 주기가 같다면 최종시한이 빠른 타스크가 높은 우선순위를 가짐
Earliest Deadline First Scheduling(EDF)	• 최종시한이 가장 임박한 타스크를 우선적으로 실행하는 방식
Priority Based Scheduling	• 각 타스크마다 실행 우선 순위를 할당하여 우선 순위가 높은 타스크를 우선적으로 실행하는 방식
Event Driven Scheduling	• 타스크를 인터럽트와 같은 Event에 동기화하여 실행하는 방식
Round Robin Scheduling	• 모든 타스크는 같은 우선순위를 가지고 CPU를 공유함 • 각 타스크마다 일정한 실행시간을 분배 • 선점형 스케줄링

웨어를 구현하는 프로그래밍 언어는 이러한 현상들을 표현하고 실행하기에 충분한 성능을 가져야 한다.

결론적으로 현재의 실시간 소프트웨어 개발 언어는 크게 C/C++ 언어와 Ada 언어로 양분된다. 물론 이들 외에도 수많은 언어가 있지만 이들 언어는 범용 소프트웨어 구현에만 유용할 뿐이다. 현재 C/C++ 언어와 Ada 언어를 제외하고는 유용한 실시간 소프트웨어 개발 툴을 가지고 있는 것은 거의 없다.

물론 Java 언어를 실시간 시스템 개발에 사용한다면 상황이 달라지겠으나 Java 언어는 시스템을 개발하기 위해 탄생한 것이 아니라 단지 여러 가지 플랫폼에서 수정 없이 동작 가능하도록 개발되었다. 현재 Java 언어는 주로 모바일 서비스 프로그래밍에 사용되고 있다.

C 언어는 본래 시스템을 개발하기 위해 태어난 프로그래밍 언어이다. C 언어는 가장 강력한 언어임에는 틀림이 없지만 여러 명의 프로그래머가 큰 시스템을 개발하기에는 부적절한 것 같다. 일반적으로 C 언어를 사용하여 50,000 라인 이상의 프로젝트를 진행하는 것은 위험도가 있다고 알려지고 있다. C 언어를 기반으로 한 객체지향 언어인 C++ 언어가 개발되었지만 조금 더 큰 시스템에 대한 지원이 좋아졌을 뿐, 대형 실시간 소프트웨어에서 구조적으로 C/C++ 언어는 쉽게 제어할 수 없는 버그를 양성할 가능성이 크다.

Ada 언어는 소프트웨어의 개발뿐만 아니라 유지, 보수에 골치를 앓던 미국방성(USA DoD)에 의해서 Pascal 언어를 기반으로 1979년에 개발되었

다. 특히 Ada 언어는 실시간 처리와 대형 내장시스템에 적합하며 특히 신뢰성이 중요한 시스템에 적당한 프로그래밍 언어이다[16]. Ada 언어는 추후 수정을 거쳐 Ada83으로 발표되고 후에 Ada95로 개정되었다[13].

미국방성이 Ada 언어를 선택하게 된 이유는 그림 1과 같이 해가 거듭될수록 하드웨어 개발 및 유지, 보수 비용보다 소프트웨어의 그것이 천정부지로 치솟고 있기 때문이었다[14]. 1986년부터 Ada 언어는 미국방성과 NATO의 중요 업무에 사용하도록 의무화되었다[16]. 예를들어 Boeing 777 비행기의 비행 소프트웨어의 99%는 Ada 언어로 개발되었다. 그렇다하더라도 Ada 언어가 실시간 시스템의 복잡성을 줄여주지는 않는다. 실시간 시스템을 이해하고 해석하고 구현하는 것은 전적으로 도구가 아닌 설계자의 몫이라는 것을 주지하여야 한다[19]. 도구는 단지 설계자의 아이디어를 표현해 주고 이것을 좀 더 쉽게 테스트할 수 있도록 도와줄 뿐 더 좋은 아이디어를 제공해 주는 것은 아니다.

개발 프로젝트 초기 단계에서 적절한 분석을 통해 알맞은 개발 규격을 도출하고 개발하고자 하는 소프트웨어의 전체 구성 및 인터페이스를 설계하는 것이 매우 중요하다. 중,대형 프로젝트에서는 기획 및 설계의 오류가 실제 코딩의 오류보다 더 심각한 영향을 미친다. 미국의 위성개발사이며 다목적위성 1호의 개발을 공동으로 수행했던 TRW의 경우를 살펴보면 소프트웨어 오류의 약 2/3는 개발 초기의 잘못된 설계로부터 발생되었고 실제로 코딩에 의한 오류는 상대적으로 적은 부분(36 %)을 차지하였다[21].

이와 더불어 소프트웨어 개발을 위한 개발 언어 선택, 개발 툴 선택, 개발 환경 구축도 매우 중대한 영향을 미친다. 이렇듯 개발 초기 단계의 작업의 품질은 그림 2에서 볼 수 있듯이 소프트웨어 전체 비용에 막대한 영향을 끼친다[15]. 왜냐하면 개발 초기의 부정확한 소프트웨어 구현에 따라서 야기되는 개발 완료 단계의 막대한 소프트웨어 디버그 비용을 감당할 수 없기 때문이다. 가령 설계단계에서 미리 문제점을 발견하여 수정하였다면 소프트웨어 조립단계에서 문제점을 수정하는 것보다 약 8배의 비용절감을 할 수 있다.

이와 같은 관점에서 접근할 때 실시간 소프트웨어 개발 언어의 선택은 가장 중요하고 근본적인 문제가 될 것이다. 미국방성은 자신들에게 납품하는 프로그램은 반드시 Ada 언어를 사용하도록 권장하고 있다. 만약 하청업자가 다른 언어를 사용한다면 이것이 별도의 성능 요구사항을 만족시키는지 확인해 주어야 한다.

표 4에는 C/C++ 언어와 Ada 언어의 특징과 장단점을 분석한 결과를 나타내었다[16].

6. 실시간 소프트웨어 개발 관리

소프트웨어를 개발하기 위해서, 특히 실시간 소프트웨어를 개발하기 위해서는 소프트웨어 개발 형상 관리(Software Development Configuration Management)의 중요성이 대두된다. 이것이 비록 실제 소프트웨어 개발의 구현론과는 거리가 있다고 하더라도 소프트웨어 프로젝트 및 소스를 관리하고 개발 팀원들 사이의 유기적인 인터페이스를 관리하는 것은 소프트웨어 개발의 효율성 증대뿐만 아니라 자칫 사장되기 쉬운 소프트웨어의 유지, 보수를 쉽게 하기 때문에 중, 대형 시스템의 소프트웨어를 개발할 때에는 CASE(Computer Aided Software Engineering) 툴과 같은 소프트웨어 개발 관리 툴의 사용도 적극 고려하여야 한다. 실시간 시스템의 문제를 해결하고 소프트웨어를 개발하는 것은 개발자의 몫이지만 개발 프로젝트를 관리하는 것은 사람이 하기에는 벅찬 일이다.

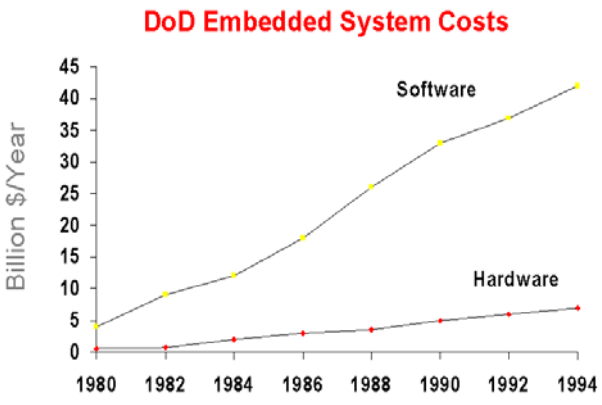

그림 1. 미국방성 소프트웨어/하드웨어 비용 대비

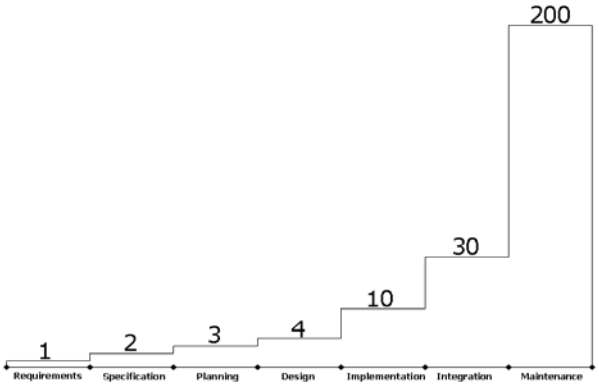

그림 2. 소프트웨어 수정에 따른 단계별 비용 대비

표 4. C/C++ 언어와 Ada 언어의 비교

	C/C++	Ada
특 징	• C - 구조화 언어 • C++ - 객체지향 언어	• Package 모듈화 • 객체지향 언어
장 점	구조적 데이터, 실시간 처리, 추상화, 빠른 수행 C++ - 예외처리	구조적 데이터, 실시간 처리, 추상화, 병행 처리, 예외 처리, 관리가 편함
단 점	• C - 관리가 불편	빠른 개발이 어려움

소프트웨어 개발 팀을 조직하는 것도 향후 소프트웨어 품질에 큰 영향을 미친다. 소프트웨어 개발 팀의 조직은 크게 책임자 중심 팀(Chief Programmer Team)과 민주적 협동 팀(Democratic Team)으로 나눌 수 있는데 두가지 조직을 병행하여 조직하는 것이 효율적이라고 생각된다.

그림 3에는 소프트웨어 개발의 일반적인 개발 사이클이 나타나 있다. 이 개발 사이클의 각 단계에서 할일을 소프트웨어 개발 기획 단계에 미리 정의하고 각 사이클에서의 결과물을 체크하여 소프트웨어 개발이 계획대로 진행될 수 있도록 관리가 필요하다.

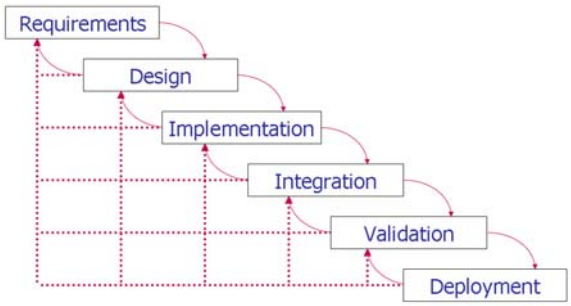

그림 3. 소프트웨어 개발 사이클

이와 더불어 개발되는 소프트웨어의 성능을 측정하고 관리하는 것도 필요하다. 가능한 적은 메모리, 전력 소모, CPU 처리, 하드웨어 장치와 설계 노력을 가지고 요구되는 요구사항을 만족시키는 능력이 소프트웨어의 성능과 직결된다. 이들 판단 근거 내에서 서로 조화를 이루게 실시간 내장형 소프트웨어를 개발하는 것이 필요하다.

실시간 소프트웨어 개발 방법론에는 다음과 같은 것들이 있다.

- RTSAD(Real-Time Structured Analysis and Design)
- JSD(Jackson System Development)
- NRL(Naval Research Laboratory Softwrae Cost Reduction Method)
- OOD(Object-Oriented Design)
- DARTS(Design Approach for Real-Time Systems)
- ADARTS(Ada based Design Approach for Real-Time Systems)
- CODARTS(Concurrent Design Approach for Real-Time Systems)

소프트웨어 개발 관리 툴에는 다음과 같은 것들이 있다.

- IBM Rational Rose
- Borland Together

소프트웨어 버전 관리 툴에는 다음과 같은 것들이 있다.

- PVCS
- CCC
- Source Integrity
- Star Team

7. 결론

실시간 시스템은 실제 물리세계의 현상을 처리하는 시스템이기 때문에 그 현상만큼이나 종류가 다양하고 각각의 특징을 가지고 있다. 따라서 물리현상을 구현하는 타스크를 정의하고 타스크 간의 또는 리소스 간의 스케줄링을 잘 설계하는 것이 실시간 소프트웨어의 개발 포인트라고 할 수 있다.

중, 대형 시스템에서는 소프트웨어의 구조적인 문제점을 파악하기가 매우 힘들기 때문에 실시간 소프트웨어의 초기 개발 계획을 수립할 때에는 개발 말기에 예상되는 소프트웨어의 수정 사항에 대비해서 관리가 용이하도록 고려가 되어야 한다.

실시간 시스템은 동시성 시스템(Concurrent System)이다. 동시성 시스템이란 시스템의 구성 요소들이 서로 상호 작용을 하는 시스템을 말한다. 구현하고자 하는 동시성은 개발하고자 하는 실시간 시스템에 따라서 매우 복잡해 질 수 있으며, 이것은 효율적이고 유연한 실시간 시스템의 초기 설계를 어렵게 한다. 단일 프로세서 아래에서만 아니라 다중 프로세서 상에서도 여전히 동시성의 문제는 설계자에게 많은 부담을 안겨주며 하드웨어의 제약 때문에 실제 응용에서는 순차적(Sequential)으로 구현이 되어야 하지만 아이러니하게도 이러한 순차적 개념은 동시성 시스템을 이해하고 설계하고 구현하는데 매우 제약을 가져온다.

따라서 구현하고자 하는 실시간 시스템을 이해하고 개발 및 관리 계획을 수립하고 실시간 시스템 구조를 설계하는데 전체 개발시간의 20 ~ 30%를 할당하는 것이 바람직하다고 생각한다.

인공위성의 제어 소프트웨어는 연성 실시간 시스템에 해당되고 로켓의 제어 소프트웨어는 강성 실시간 시스템이라고 볼 수 있지만 개발 프로세스는 동일하다고 본다.

프로그래밍 언어의 선택은 매우 중요한데 해외 대부분의 위성 제작사들은 Ada 언어를 사용하여 위성 소프트웨어를 개발하고 있었다. 이것은 단적으로 프로그래밍 언어의 관리성이 소프트웨어의 품질에 얼마나 많은 영향을 주는지 보여주는 증거라고 생각한다. Ada 언어는 엄격한 문법을 가지고 있고 다수의 프로그래머가 같이 작업하는데 따르는 오류의 가능성을 최소화할 수 있다.

위성이나 로켓의 내장 실시간 소프트웨어를 개발하기 위해 C/C++언어와 Ada 언어 중 어느 것을 사용하는 것이 좋은지에 대한 결정은 신중한 검토를 통해서 결정되어야 할 것 같다. 국내에서 Ada 언어는 많이 사용되지 않은 이력을 가지고 있고 무엇보다 인적자원이 부족한 것이 현실이다. 그렇다면 Ada 언어로 구현된 시스템을 유지 보수하는 것은 때로 어려운 과제로 남을 수도 있다.

그리고 C++ 언어의 대부분의 기능은 내장형 시스템에 대해서는 불필요한 기능이다. 비록 이것이 기능 자체에 대해서는 훌륭하다고 하더라도 이런 기능의 사용은 성능이 극도로 최적화되어 있는 실시간 내장형 시스템에 무리한 오버헤드를 필연적으로 준다.

하지만 C++ 언어는 C 언어가 가지고 있지 않은 소프트웨어 관리성에 대한 이점을 제공하고 있다. C 언어는 관리성만 보강되면 내장형 소프트웨어 개발에 최적의 언어라고 볼 수 있다.

신뢰성있는 실시간 소프트웨어를 개발하기 위해서는 먼저 효과적인 개발과 협동이 이루어질 수 있는 개발팀을 조직하는 것이 선결과제이고 그 후 구현하고자 하는 실시간 시스템을 이해하고 구현론을 수립하고 철저한 소프트웨어 형상관리를 통해서 체계적으로 개발하여야 한다.

참고문헌

1. http://www.faqs.org/faqs/realtime-computing/faq/
2. Herman Bruyninckx, "Real-Time and Embedded Guide", K.U.Leuven, Mechanical Engineering
3. Jack G. Ganssle, "The Art of Designing Embedded Systems", Newness
4. C. M. Krishna, Kang G. Shin, "Real-Time Systems", McGrawHill
5. Hermann Kopetz, "Real-Time Systems - Design Principles for Distributed Embedded Application", Kluwer Academic Publishers
6. Giorgio C. Buttazzo, "Hard Real-Time Computing Systems - Predictable Scheduling Algorithms and Applications", Kluwer Academic Publishers
7. Loic P. Briand, Daniel M. Roy, "Meeting Deadlines in Hard Real-Time Systems - The Rate Monotonic Approach", IEEE COMPUTER SOCIETY
8. Phillip Laplante, "Real-Time Systems Design and Analysis - An Engineer's Handbook", IEEE COMPUTER SOCIETY
9. Edward A. Lee, "Embedded Software", Vol 56, Academic Press, London, 2002
10. Stephen A. Edward, "Design Languages for Embedded Systems", Columbia University, NY, 2003
11. Neil C. Audsley, "Deadline-Monotonic Scheduling", York University, York

12. Sunondo Ghosh, Rami Melhem, Daniel Mosse, "Fault-Tolerant Rate-Monotonic Scheduling", Pittsburgh University, Pittsburgh
13. Michael B. Feldman, "Ada95 In Context", Handbook of Programming Languages, vol. 1, Macmillan, 1998
14. Mani Srivastava, "Embedded Computing Systems:An Overview", EE202A, UCLA - EE Department, 2003
15. CS 599, "Overview of Software Engineering Principles", http://sunset.usc.edu/classes/cs599_2002/
16. Steve McConnell, "Code Complete", Microsoft PRESS
17. Raj Kamal, "Embedded Systems - Architecture, Programming and Design", Tata McGraw-Hill
18. John Sidney Davis II, "Order and Containment in Concurrent System Design", California University, Berkeley
19. D. Atiya, S. King, "Communicating Ada Tasks", Computer Science Department, York University
20. J. S. Briggs, et al, "Debugging Distributed Ada Programs", Real-Time and Distributed Systems Research Group, Computer Science Department, York University
21. James Martin, Carma McClure, "Structured Techniques for Computing", Ch 28, Prentice-Hall, 1985